D-BRIDEMAID: A Distributed Framework for Collaborative and Dynamic Analysis of Android Malware

Antonio La Marra, Fabio Martinelli, Francesco Mercaldo*, Andrea Saracino, and Mina Sheikhalishahi
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
name.surname@iit.cnr.it

Received: January 6, 2020; Accepted: July 10, 2020; Published: September 30, 2020

Abstract

Android malware are currently the only practical vector to bring security attacks to smartphone and tablets. Malware detection and prevention of zero day attacks requires a prompt analysis, which would benefit in terms of timeliness and accuracy, from being collaborative. This paper presents D-BRIDEMAID a reputation-based framework able to analyse Android applications, with the aim to exploit an hybrid static/dynamic framework for malware analysis to initiate a distributed app evaluation, involving real users willing to test the security features of an app on their device. This work focuses on the definition of the collaborative protocol, the reputation based incentive system and the models to compute revenue for users and security of apps. Simulative and real world experiments are presented to validate the model.

Keywords: mobile security, collaborative analysis, dynamic analysis, Android

1 Introduction

Android malicious applications (apps) are growing in last years [1]. As a matter of fact, 98% of mobile malware targets Android environment [2]: in 2016 every 4.6 seconds a new malware payload is emerging, while in the first quarter of 2017 this only takes 4.2 seconds [3]. Over one million of mobile malware apps is currently distributed in the wild, showing always new attack vectors, together with several aggressive techniques able to hide the malicious payload and cloaking its effect, deceiving both users and current antimalware technologies [4]. Early detection of Android malicious payloads requires an active monitoring of the main channels used for app distribution, and rigorous app analysis which should not be limited to the standard, easy-to-deceive signature matching. In fact, in order to maximize the possibility of detecting unknown threats i.e., the so-called zero-day attacks, the detection method should consider static [5] and dynamic analysis techniques [6, 7], basically based on behavioral analysis, hardly deceived by widespread and common obfuscation techniques employed by malware writer with the aim to elude the signature-based detection [8] provided by common antimalware technologies. However, this analysis process is computationally heavy and time consuming, hence difficult to apply on a large scale as a standalone service. This consideration is further worsened by the fact that often dynamic still automatic analysis is not effective, due to the ability of malware payloads to recognize when they are runned inside an automated testing environment such, for instance, emulators and sandbox [9]. This is the reason why app analysis might require direct interaction with real world users, which maximize the needed effort and

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 11(3):1-28, Sept. 2020
DOI:*10.22667/JOWUA.2020.09.30.001*

*Corresponding author: Università degli Studi del Molise, Via Francesco De Sanctis, 1, 86100 Campobasso, Italy, Tel: +39-0874-4041

can make the malware analysis for early detection of zero day threats, unfeasible on a large scale. We argue that for a possible solution, it is necessary to design a collaborative approach [10], which leverages on real device users, with a modest security awareness, with the aim to analyze several malware samples in a shorter amount of time, still able to handle the introduced issues on trust and reliability.

Starting from these considerations, in this paper we propose D-BRIDEMAID (Distributed BRIDEMAID) a distributed framework to enable the cooperative analysis of Android apps , aimed at early detection of new threats through distributed dynamic analysis. The framework leverages on BRIDEMAID (Behavior-based Rapid Identifier Detector and Eliminator of Malware for AndroID), an accurate software for malware detection based on static and dynamic analysis, which obtains an accuracy equal to 99.7% on a dataset of more than 3k real world malicious apps. This extension of the framework proposed in [6], is designed to automatically analyze the behavior of apps on the devices of a pool of users, collecting their reports and deciding on the trustworthiness of the tested app. These users, willing accept to participate to a collaborative program for malware detection in which they install on their devices new apps, potentially malicious, where the BRIDEMAID analysis software is running. Based on the observed behaviors and eventually raised alerts, the user can send a report for every evaluated apps, deeming it as *Genuine* or *Malicious*. The report of different users for the same app are collected in order to assess if an app is malicious or genuine, and weighted according to parameters concerning the user reputation and the reliability of report, based on analysis time and additional parameters which will be discussed in the following. Since the envisioned model is Peer-to-Peer (P2P), without hypothesis of trust on users, the proposed framework exploits a reputation management algorithm to detect and tackle eventual attackers, not considering their reports, once they have been detected.

The main contribution of the paper are the following:

- We design a distributed framework for analysis of Android malicious applications which exploits a set of cooperating users and advanced techniques for monitoring and detect malicious behaviors based on static and dynamic analysis.
- We propose a mathematical model with the aim to evaluate the reliability of different reports and combining them into a single index which expresses the overall reliability of an app evaluation.
- We report a set of features relevant in the evaluation of the reliability of an app testing, also related with specific behaviors.
- We introduce a probabilistic model for incentivizing users in participating and behaving correctly and with the aim to compute the correct revenue value.
- A set of simulative experiments is reported in order to evaluate the framework and mathematical model we propose, measuring its reliability at the variation of the number and type of attackers, considering four different attacker models and the possibility of colluding users.
- Finally we evaluate the proposed framework through a set of real world experiments involving a limited group of real users, collaboratively testing real apps on real devices in order to demonstrate that the effectiveness of the proposed solution in the real world environment.

This paper extends the work presented in [11] by adding a new incentive model related to the data sharing framework (Section 4), a more complicated attack model for new simulative experiment which also considers colluding attackers (Section 5) and a set of real world experiments with real users and real devices using real apps. The remainder of the paper is organized as follows. Section 2 reports the concept of static, meta data, and dynamic analyses of Android apps better explored in [6] and [8]. Section 3 describes the proposed framework, attacker model, and the workflow. Also it introduces the

theoretical aspects of reliability, users' reputation and result's validity calculation. Section 4 presents all the theoretical aspects of the incentive-based mechanism for information sharing. Section 5 validates our framework showing a set of simulative experimental results with four different attacker and several configurations. Section 6 illustrates the real world experiments performed. Section 7 reports a list of related work in order to explain the current state of the art related to the malware detection in mobile environment, while Section 8 concludes the paper.

2 Background

In this section we describe the BRIDEMAID approach i.e., the on device analysis framework for Android applications able to combine static and dynamic analysis in order to discriminate between malware applications and legitimate ones. The proposed framework exploits a multi-level and multi-feature analysis including permission scoring and evaluation, opcode analysis, kernel level monitoring and API calls hijacking. Further details can be found in [6].

2.1 Static Analysis

With regards to static analysis, we take into account a binary classification problem in which an unknown application has to be classified as malicious or legitimate. The static analysis phase consists of two phases; (i) a learning phase: in this phase the classifier is trained using a labelled dataset of applications, and (ii) the classification phase: in this phase an input application is classified as malicious or legitimate. We consider as feature *n*grams of smali code with $n = 2$ because a previous work [12] demonstrated that the sequences of two consecutive opcodes exhibit better performance in Android malware detection. Furthermore, BRIDEMAID exploits an analysis of meta-data gathered from the manifest file in order to analyze the permissions required by the application under analysis with the aim to compute a threat score. Furthermore, we consider the *market* of provenance, the *download number*, the *user rating* and the *developer reputation* (if available). All these parameters are combined through the Analytic Hierarchy Process [13], able to decide whether the application under analysis should be labelled as *Trusted* or *Suspicious*. Mobile applications which are considered trusted can be executed on the device without additional check on the performed behaviors. Applications which instead are classified as suspicious will be subject to the control of the dynamic analysis monitors, as discussed in the following.

2.2 Dynamic Analysis

The dynamic analysis, considers both global features, i.e. related to the device and operative system, and local ones which are related to specifically monitored application under analysis. This analysis is based on two core elements that monitor different sets of features: the Global Monitor and the Per-App Monitor. The Global Monitor is able to observe the device and OS features at three different levels, i.e. kernel (*SysCall Monitor*), user (*User Activity Monitor*) and application (*Message Monitor*). These features are monitored regardless of the specific app or system components generating them, and they are used in order to shape the current behavior of the device itself: these behaviors are classified as *genuine* or *malicious* by a classifier. The Per-App Monitor, implements a set of *known behavioral patterns* with the aim to monitor the actions performed by the set of suspicious apps.

3 Architecture and Workflow

In following section we explain the architecture of D-BRIDEMAID, introducing the components, their interactions and the operative workflow.

3.1 System Model

Figure 1 depicts the main components of the envisioned architecture. As shown, the apps are stored

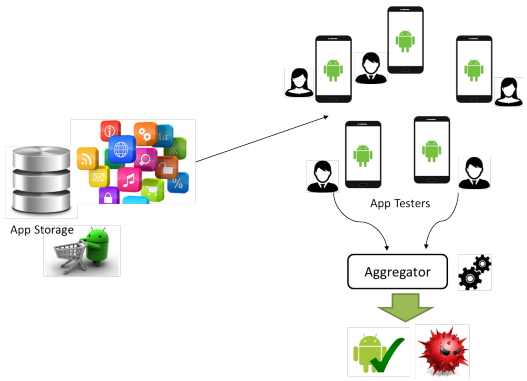


Figure 1: Architecture of the proposed framework.

in a cloud storage which also acts as orchestrator for the whole framework. The main actors of the framework are the *app testers*, which are users that agree to participate to this app evaluation process. Tester devices are all equipped with D-BRIDEMAID, which acts as a dynamic Intrusion detection System (IDS), reporting to the user suspicious behaviors. The testers willing offer their devices and their app interaction time, being aware that both their mobile devices and the contained information might be exposed to the risk of malicious apps. This risk is strongly mitigated by the effectiveness of BRIDEMAID, which detects and stops malicious identified behaviors. Moreover, testers receive a reward for the service they are offering and the risk they are taking. The reward, which could also be monetary-based, is used as incentive for user participation and together with a reputation-based algorithm discussed in the following, fosters a correct user behavior. After evaluating an app, the user submits a report, based on eventual BRIDEMAID alerts, deeming the tested app either as malicious or genuine. Reports of testers concerning every analyzed app are then collected by an *Aggregator* which gets a decision on the app trustworthiness, out of all the received reports.

3.2 On Device App

The D-BRIDEMAID framework is composed by an host application to install on the tester device. It is a lightweight application and does not hinder the everyday smartphone usage.

Once installed, the D-BRIDEMAID host application authenticates the tester through the IMEI and the IMSI in order to ensure that the application to test is running using a real environment and not on an emulated one (we check this because mobile malware usually do not perform malicious action if executed on emulated enviroment, this happens usually to elude honeypot. The D-BRIDEMAID application, once the user is authenthicate to the framework, proposes a list of unknown applications to test. The D-BRIDEMAID application shows, for each application, the results derived from static analysis module, and the tester will decide whether run the application. In this case D-DRIDEMAID will install and launch the application to test on the device. Once the application to test is running, D-BRIDEMAID is

able to check the background or foreground time of application, these parameters are useful to compute the user reliability, as explained in the next section.

D-BRIDEMAID host application is also capable to intercept a set of legitimate events that tipically stimulate a malicious payload. An event in Android is sent from an application (i.e., the gps localization, the presence of WiFi newtorks) and from the device user (i.e., boot the device, charge the phone).

Table 1 shows the most occurring events able to activate the malicious behavior, which the percentage that the considered event is able to trigger the malicious payload. As matter of fact, there are several ways that a malware may employ to be activated, each one associated with an activating system event. This list and the percentage of malicious payload activation has been compiled taking into account the events which most frequently trigger the payload in Android malware, according to several studies [14].

Table 1: Events used in order to trigger the malicious payload.

#	Event	Description	App Behavior	User Behavior
1	*BOOT*	boot completed	80%	50%
2	*CALL*	incoming call	30%	75%
3	*SYS*	phone rooted	30%	35%
4	*BATT*	battery status change	50%	85%
5	*SMS*	reception of SMS	70%	85%
6	*NET*	connectivity change	50%	80%

In Table 1, the first row represents the `BOOT` event, which is the most used within existing Android malware. This event will be triggered and sent to all the applications installed in an Android device as the system finishes its booting process, a perfect timing for a malware to kick off its malicious services. By listening to this event, the malicious payload can activate itself without user's interventions or interactions with the system. Another event used from malware writers is the `CALL` one (second row in Table 1) event: this event will be sent in broadcast to the whole system (and all the running applications) when a new CALL is being received. The `SYS` events is accepted by rooted device, as matter of fact due to their open-source nature, many users are able to root their device in order to customize it and to expand their functionality. The `BATT` event is triggered when the power is connected or disconnected and when the operating system send the battery low and battery ok signals. The `SMS` event is transmitted to the system when a SMS message is received. Through this event, the malicious payload has the ability to respond to specific incoming messages with the aim to undertake malicious actions. The last event, the `NET` one, is transmitted when a change in the data connection happens (for instance when the connection switches from GPRS to HSDPA network).

3.3 Workflow

Being a distributed system, D-BRIDEMAID needs to define a workflow to synchronize the actions of all users testing apps. This workflow must be designed in the direction of flexibility, to ensure the highest possible participation for each analysis, which might also include the possibility of having the same user testing at the same time two different apps. Duration of app testing should be long enough to maximize the probability that eventual malicious code activates, still limited to allow a timely decision on the app trustworthiness.

We assume the system to own at each time a set of n active users (testers), i.e. currently able to install and test the functionality of apps. Apps to be tested are chosen by D-BRIDEMAID from a repository of apps considered suspicious after a first static analysis done by the system exploiting opcode n-grams and app metadata. D-BRIDEMAID chooses a short list of apps, proposing them to the n users. D-BRIDEMAID can decide which apps to propose to each user, in order to balance the number of users testing each app. Each user can choose among the proposed ones, the apps to test. Once a user accepts, the app is downloaded and installed on the device and the test can be run for a configurable time span

named *evaluation round*, of duration t_r. From the time the first user installs the app, any other invited user can start its evaluation round, given that accepts the invitation before the evaluation round of the first user ends. The duration of the evaluation round is the same for each user, whilst the whole duration of an app evaluation is named *evaluation epoch*, having a variable duration named t_e. Due to the constraints on the joining time, it is ensured that the duration of an evaluation epoch cannot be longer than twice the evaluation round, i.e. $t_r \leq t_e \leq 2t_r$. This workflow is depicted in Figure 2, showing an example in which four testers join to an evaluation epoch. The event of joining is represented by the small arrows. As shown, after the end of the first evaluation round, no other testers can join to the evaluation. At

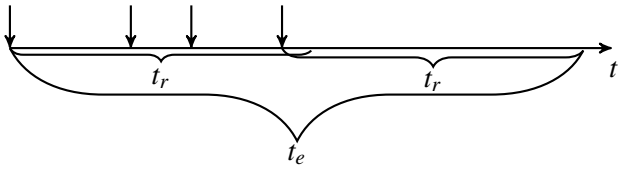


Figure 2: Evaluation epoch timeline.

the end of her own evaluation round, the user produces a *report*. The report contains the following information: a binary decision to classify the app as malicious or genuine. At the end of the evaluation epoch, the aggregator collects all the reports concerning the analyzed app and computes a decision based on the aggregated results. After the decision has been taken, tester reputation is updated according to Algorithm 1 and revenue is assigned to users providing a useful, i.e. reliable and correct result.

As additional notes, the service is able to handle parallel session, with users testing on the same device different apps at the same time. However, for each app there is a specific evaluation epoch, not necessarily synchronized with the evaluation epochs of the other apps.

3.4 Reliability Computation

This section defines the *reliability* of a tester report as a function, which returns a real number in $[0,1]$. To shape the reliability function, the following components are considered:
• *BRIDEMAID report*: if a user determines that an application α is malicious and she declares that her evaluation result is based on BRIDEMAID report on her device, then the provided information of this user should receive the highest *reliability* score. This assumption is considered sound thanks to the high accuracy of the BRIDEMAID software described in 2.
• *Time*: if the app is considered genuine, an higher reliability score is given if the app has been in background or foreground for a time as close as possible to t_r.
• *Events*: additional reliability is assigned to report if one or more of the events in Table 1 is performed.
In the following, we detail the aforementioned components and their impacts on reliability score.

Let user u_i report that the application α is malicious, at time t. We denote it as $f_i(\alpha,t) = 1$, where f_i is a *Boolean* function which returns 0 if α is *genuine* or 1 if *malicious*. Formally:

$$f_i(\alpha,t) = \begin{cases} 1 & u_i \textit{ reports } \alpha \textit{ is malicious at } t \\ 0 & u_i \textit{ reports } \alpha \textit{ is genuine at } t \end{cases}$$

BRIDEMAID Report If user u_i reports the application α as malicious, then the maximum *reliability* score should be returned. Formally, we define:

$$f_i(\alpha,t,\mathcal{B}) = \begin{cases} 1 & u_i \text{ reports } \alpha \text{ as malicious} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Time Let t_0 be the beginning of the first evaluation round for α. We denote by t the amount of time passed from t_0; by T the maximum time duration, i.e. equivalent to *evaluation epoch*; and $t_i(\alpha,t)$ the duration of time that application α has been active in background or foreground for user u_i'. Hence, $t_i(\alpha,t) \in [0,T]$.
Moreover, the time should affect the reliability in exponential ascending scheme, since the probability of an app activating the malicious payload is high at the beginning and exponentially decreases during time. Therefore, we define the following function for computing the impact of time in *reliability* of the report of user u_i about app α:

$$\tau_i(\alpha,t) = 1 - e^{-t_i(\alpha,t)} \qquad t_i(\alpha,t) \in [0,T] \tag{2}$$

Events Representing by $\mathcal{B}_i(t)$, $\mathcal{C}_i(t)$, $\mathcal{Y}_i(t)$, $\mathcal{M}_i(t)$, $\mathcal{A}_i(t)$, $\mathcal{S}_i(t)$ and $\mathcal{N}_i(t)$ the Boolean functions of *events* reported in Table 1 respectively, reported by user u_i at time t, such that it equals to 1 if user u_i has executed the associated event during $t_i(\alpha,t)$, and 0 otherwise. Formally, let $\gamma_i(t) \in \{\mathcal{B}_i(t), \mathcal{C}_i(t), \mathcal{Y}_i(t), \mathcal{M}_i(t), \mathcal{A}_i(t), \mathcal{S}_i(t)\}$, then we have:

$$\gamma_i(t) = \begin{cases} 1 & u_i \textit{ has executed } \gamma_i \textit{ till } t \\ 0 & u_i \textit{ has not executed } \gamma_i \textit{ til } t \end{cases}$$

As shown in Table 1, different events have different probability to be arisen. Hence, we consider the following *weights* indicating the impact of each event execution:

$$\omega_{\mathcal{B}} = \frac{0.8}{3.6},\ \omega_{\mathcal{C}} = \frac{0.3}{3.6},\ \omega_{\mathcal{Y}} = \frac{0.4}{3.6}$$
$$\omega_{\mathcal{M}} = \frac{0.4}{3.6},\ \omega_{\mathcal{A}} = \frac{0.5}{3.6},\ \omega_{\mathcal{S}} = \frac{0.7}{3.6},\ \omega_{\mathcal{N}} = \frac{0.5}{3.6}$$

Eventually, the effect of events in reliability reported by user u_i at time t, denoted by $\mathscr{E}_i(\alpha,t)$, is computed as follows:

$$\begin{aligned}\mathscr{E}_i(\alpha,t) =& \omega_{\mathcal{B}} \cdot \mathcal{B}_i(t) + \omega_{\mathcal{C}} \cdot \mathcal{C}_i(t) + \omega_{\mathcal{Y}} \cdot \mathcal{Y}_i(t) + \\ & \omega_{\mathcal{M}} \cdot \mathcal{M}_i(t) + \omega_{\mathcal{A}} \cdot \mathcal{A}_i(t) + \omega_{\mathcal{S}} \cdot \mathcal{S}_i(t) + \omega_{\mathcal{N}} \cdot \mathcal{N}_i(t)\end{aligned} \tag{3}$$

where $\mathscr{E}_i(\alpha,t) \in [0,1]$, and the higher score shows that the report of user u_i about application α is more reliable in terms of *events* evaluation.

Having the above functions and relations, is possible to define the *reliability* function, denoted by $\mathscr{R}_i(\alpha,t)$, which returns a number in the interval $[0,1]$, such that the higher score means that the higher reliability is guaranteed on the report of user u_i about app α at time t.
Formally, we have:

$$\mathscr{R}_i(\alpha,t) = \begin{cases} 1 & f_i(\alpha,t,\mathcal{P}) = 1 \\ ((1-c)\cdot \mathscr{E}_i(\alpha,t) + c) \times \tau_i(\alpha,t) & \textit{otherwise} \end{cases}$$

where $f_i(\alpha,t,\mathcal{P})$, $\tau_i(\alpha,t)$, and $\mathscr{E}_i(\alpha,t)$ are obtained applying relations 1, 2, and 3, respectively; and c is a constant real number in $[0,1]$, where the higher number means that the more weight is attributed to the impact of time without executing the events. In our experiments, we set $c = 0.5$. The amount of reliability equals to maximum output, i.e. 1, when the user report is based on the BRIDEMAID evaluation on her device. The reliability approaches to zero if the time duration of having application α being active in the background of user u_i's device goes to zero.

3.5 Attacker Model

As the system is distributed, with testers being all peers, the lack of a root of trust exposes the framework to a set of attacks which have to be counteracted. We assume that attackers are able to completely modify the decision of BRIDEMAID, being thus able to submit their own decision for the app, with an arbitrary level of reliability. Being interested in pushing the system to accept their decisions, attackers will always choose a reliability score which is higher than the reliability threshold set by D-BRIDEMAID. A set of envisioned attackers is briefly presented in the following:
Reputation Tamperer: This malicious tester aims at tampering the reputation of one or more genuine apps, in order to discourage users from downloading it. Reports submitted by this tester for genuine apps will deem the app as malicious with an high reliability level (i.e., higher than 0.8).
Malicious App Preacher: This tester accepts to participate to the evaluation program to push the system in considering as genuine an app which is malicious. This tester when submits reports for malicious apps, will always report the app as genuine, with an high reliability level.
Coin Flipper: The coin flipper gives random decisions on the trustworthiness of an app, with the objective of damaging the system still keeping a low profile to not be easily identified. This behavior can also be used to model users with a malfunctioning BRIDEMAID app.
Persistent Liar: This tester aims at maximizing the damage to the system, performing correct and reliable app analysis but producing always the opposite decision. This attacker is particularly dangerous if colluding with other attackers showing the same behavior.

3.6 Reputation Handling

The reputation model we considered in thefollowing work is the one based on the Jøsang model depicted in [15]. The proposed model exploits a reputation score weighted by three different components i.e., *belief*, *disbelief* and *uncertainty*. The rationale behind choosing the considered model is the correspondence with the three possible actions that an UCS is able to perform when asked to provide a cached value. Every UCS has at the beginning a starting reputation score, defined as r_0. At each attribute reading in which a specific usage control system is involved, its reputation is updated according with the following formula: $r(t) = b(t) - d(t) - u(t)$, where t is the time that the reputation is requested.
Afterward, on the base of the provided value, after the decision process performed at step four of the system evolution, the three reputation component are updated according to the depicted algorithm.

The belief component is increased every time a considered user provides a value that is not wrong (the provided value is the one that is effectively considered good by the system). Whether the provided value is considered not reliable enough (less than 0.5), the uncertainty component in this case is increased, whilst the disbelief component is increased if the provided value is considered reliable but wrong. No reputation changes happen for users that choose to not provide any value. The values of Δ_d, Δ_u and Δ_b are configurable parameters. Realting to the experiment we performed in following work, the considered values are: $\Delta_b = 0.25$, $\Delta_u = 0.15$, $\Delta_d = 0.6$, whilst the acceptance threshold for reputation θ_r is equal to 0.5. These values mildly increase at each reading the reputation of those users providing correct values, strongly penalize the users providing a malicious value, in this case the aim is to immediately reduce their reputation under the acceptance threshold. We highlight that the uncertainty has a small impact on the reputation, as a matter of fact it becomes consistent only after several *non-enough-reliable* readings.

3.7 Result Validity

With the definitions of reliability and reputation, it is now possible to formally define the decision process, used by our architecture to choose the boolean value $V \in [0,1]$ for an application α based on the reliability of what the users $u_1, u_2, \ldots, u_N$ provide about this application and considering their reputa-

Algorithm 1 Updating reputation r_l

$r_l = b_l - d_l - u_l$
$b_l + d_l + u_l = 1$
for all $u \in U^j$ **do**
  if u provides a correct value **then**
    $b_l = b_l + \Delta_b$
    $u_l = u_l - \frac{\Delta_b}{2}$
    $d_l = d_l - \frac{\Delta_b}{2}$
  else
    if u provides a non enough reliable value **then**
      $u_l = u_l + \Delta_u$
      $b_l = b_l - \Delta_u$
    end if
  else
    if u provides a malicious value **then**
      $d_l = d_l + \Delta_d$
      $b_l = b_l - \frac{\Delta_d}{2}$
      $u_l = u_l - \frac{\Delta_d}{2}$
    end if
  end if
end for

tion. The higher score of *validity* means that the associated application is more probable to be malicious. Moreover, let $\theta_{\mathscr{R}}$ and θ_r be the thresholds of *reliability* and *reputation*, respectively.

To make the final decision, i.e. to give a *malicious score* to an application, the system collects the triples $U_i(\alpha,t) = (f_i(\alpha,t), \mathscr{R}_i(\alpha,\tau_i), r_i(t))$ for each user u_i who participated in the application evaluation, $f_i(\alpha,t)$ is the value provided by user u_i about application α at time t, $\mathscr{R}_i(\alpha,\tau_i)$ returns the reliability of $f_i(\alpha,t)$, and $r_i(t)$ is the reputation of user u_i at time t.
The system discards the triples for which either $\mathscr{R}_i(\alpha,\tau_i) \leq \theta_{\mathscr{R}}$ or $r_j(t) \leq \theta_r$. After discarding not reliable values, the set of n triples are considered as the following:

$$U_i(\alpha,t) = (f_i(\alpha,t), \mathscr{R}_i(\alpha,t), r_i(t)) \qquad for \quad 1 \leq i \leq n \tag{4}$$

Then, from the collected information at time t, the *validity* of application α, i.e. the malicious score of application α based on the reports of n users, denoted by $\mathcal{V}(\alpha,t)$, is computed as the following:

$$\mathcal{V}(\alpha,t) = \frac{1}{n}\sum_{i=1}^{n} f_i(\alpha,t) \times (\frac{\mathscr{R}_i(\alpha,t) + r_i(t)}{2}) \tag{5}$$

where the higher output of $\mathcal{V}(\alpha,t)$ shows the higher malicious score of application α.
In this work, if $\mathcal{V}(\alpha,t) \geq 0.55$, then the system reports the application α as malicious. Moreover, if the number of reliable reports, i.e. reporting a reliability higher than $\theta_{\mathscr{R}}$ is lower than a configurable percentage of participating users, the evaluation is not considered valid and the app is scheduled for reevaluation.

4 Incentive-based Design of Data Sharing Framework

Participating in framework generally causes the user various costs, for instance mobile device battery energy cost, charges asked by the operator for bandwidth needed in order to transmit data, processing power cost, or discomfort of the user resulted from manual effort to submit data.

For survivability of the proposed system, it is crucial to have an appropriate *incentive mechanism* in order to motivate users to participate in data collection. It is assumed that the participation cost is private information for users, and they are strongly motivated to misreport the actual cost to receive higher revenue.

The incentive mechanism should be designed correctly such that it addresses the following challenges:

1) The mechanism should motivates the user to participate. This means that each user should obtain *utility* at least as much as not participating.

2) Since the service does not know the actual participation cost, as it is expected by users, participants have strong intent to misreport their cost, i.e. expressing a higher cost rather than the actual one for obtaining higher *revenues*.

3) The quality of information that a user provides should be considered definitely in the *participation level*, i.e. for a user who provides always low quality information, it should be given little or no compensation.

4) The *quality of service* should be guaranteed by the participation levels of data providers, while at the same time the expenses of the service provider needs to be minimized.

4.1 System Model

Let the set $\mathcal{N}$ of n users provide reliable information as explained in Section 3.4. Suppose that p_i denotes the payment from service provider to user u_i. Then, the main components of the system are the following:

1) *Participation level:* The participation level of user u_i at current time t, denoted by $l_i(t)$, is the average level rate, i.e. the average rate with which the user submits her report to the provider. In our system, lets that from the time that user u_i has joined to the system till the current time t, $N_i(t)$ times the system proposed her an app for evaluation, and suppose $n_i(t)(\leq N_i(t)$ times the user u_i answered to this request. Thence, the participation level of user u_i at time t is computed as $\frac{n_i(t)}{N_i(t)}$. For the sake of simplicity, we use l_i instead of $l_i(t)$, while always it is meant the participation level of user u_i at current time t.

2) *Participation cost:* Each user has a perceived cost $C_i > 0$ per unit of participation level. For example the perceived energy cost depends on proximity of the mobile device for accessing to wireless and on the level of battery energy. The power cost depends on the number of jobs processed by the device processor. The cost of manual data insertion is dependent to several factors, such as the dissatisfaction of user u_i to participate in data sharing. This dissatisfaction can be resulted from device resource consumption, or time, attention and effort which needs to be put by the user.

Suppose that the cost C_i for user u_i be a continuous random variable which takes value in the range $[c^i_{min}, c^i_{max}]$. Since C_i is private information of user u_i, thence C_i's are independent variables. Moreover, let $h_i(.)$ denote the probability density function of C_i, and $H_i(.)$ be the corresponding cumulative density function. Then, the *utility* of user u_i for participation level l_i and payment p_i is given by:

$$\mathcal{U}_i = p_i - C_i \cdot l_i$$

The quality of service depends on (1) the participation level l_i of each user u_i and (2) the quality of submitted information by each user. To measure the quality of submitted information, let $U_i(\alpha_z,t) = (f_i(\alpha_z,t), \mathscr{R}_i(\alpha_z,\tau_i), r_i(t))$ be the information of user u_i about application α_z, for $1 \leq z \leq Z$, at time epoch

τ prior to t, then the *quality* indicator for user u_i at time t can be computed as follows:

$$Q_i(t) = \frac{1}{Z}\sum_{z=1}^{Z}\left(\frac{\mathscr{R}_i(\alpha_z,\tau_i)+r_i(t)}{2}\right) \tag{6}$$

The quality of service is dependent to two factors, namely participation level and quality of data. More precisely, let $L(t) = (l_1,\ldots,l_n)$ be the vector of participation levels, and $Q(t) = (Q_1(t),\ldots,Q_n(t))$ be the vector of data quality, where l_i and $Q_i(t)$ are participation level and the data quality of user u_i $(1 \leq i \leq n)$, respectively. Thence, we denote by $g(L(t),Q(t))$ the quality of service at time t, and compute it as the following:

$$g(L(t),Q(t)) = \frac{1}{n}L(t)\times Q(t) = \frac{1}{n}\sum_{i=1}^{n} l_i \cdot Q_i(t) \tag{7}$$

We denote by θ_Q the level of acceptable quality of service for the subscribers. Hence, the system needs to operate under the constraint $g(L(t),Q(t)) \geq \theta_Q$.

4.2 The mechanism

Upon receiving the request for participation, the users report their perceived cost per participation level. This cost is the minimum compensation requested for participation. The service collects users "*declared cost*" vector $\mathcal{C} = (c_1,\ldots,c_n)$, where c_i represents the declared cost of user u_i.

An incentive-based mechanism $M(\mathcal{C})$ for finding a balance between users' participation and system payment, consists of computing *participation level vector* $L(\mathcal{C}) = (l_i(C) : i \in N)$ and *payment level vector* $P(\mathcal{C}) = (p_i(C) : i \in N)$, i.e. $M(\mathcal{C}) = (L(\mathcal{C}),P(\mathcal{C}))$. Note the dependence of participation level vector $L(\mathcal{C})$ and payment level vector $P(\mathcal{C})$ for each user u_i on the entire vector of declared cost $\mathcal{C}$. To compute optimally these vectors, in what follows we first present some basic notations of the appropriate incentive-based mechanism. Afterwards, in Section 4.3 we present the optimal solutions of $L(\mathcal{C})$ and $P(\mathcal{C})$ for the mechanism $M(\mathcal{C})$.

4.2.1 Bayesian Game

In game theory, a *Bayesian game* is a game in which the players have incomplete information about other players (for instance, their available strategies or payoffs), but they have beliefs with the known probability distribution [16, 17, 18]. In our game, each user u_i knows only her own cost c_i and has probabilistic knowledge about the costs of others. The costs of the others is denoted by $\mathcal{C}_{-i} = (c_j : u_j \in \mathcal{N}, u_j \neq u_i)$. Each user u_i tries to maximize its *expected utility*

$$\mathbb{E}_{\mathcal{C}_{-i}}[\mathcal{U}_i(\mathcal{C})] = \mathbb{E}_{\mathcal{C}_{-i}}[p_i(\mathcal{C}) - c_i \cdot l_i(\mathcal{C})] \tag{8}$$

where the expectation is taken with respect to the *types* of other users. A declared cost vector Y^* is *Bayesian Nash equilibrium* if for each user $u_i \in \mathcal{N}$, we have:

$$\mathbb{E}_{Y^*_{-i}}[\mathcal{U}_i(y_i^*,Y^*_{-i})] \geq \mathbb{E}_{Y^*_{-i}}[\mathcal{U}_i(y_i,Y^*_{-i})] \tag{9}$$

for all $y_i \in C_i, y_i \neq y_i^*$. This means that in Bayesian Nash equilibrium, no user has incentive for changing its cost declaration, due to the fact that such a change would not lead to the higher utility.

4.2.2 Incentive Compatibility

A mechanism is called *incentive compatible* (IC), whether the strategy where each user reports its true cost is a Bayesian Nash equilibrium, i.e. for each user u_i, we have:

$$\mathbb{E}_{\mathcal{C}_{-i}}[p_i(\mathcal{C}) - c_i \cdot l_i(\mathcal{C})] \geq \mathbb{E}_{\mathcal{C}_{-i}}[p_i(y_i, \mathcal{C}_{-i}) - c_i \cdot l_i(y_i, \mathcal{C}_i)] \tag{10}$$

for all $y_i \in C_i$ that $y_i \neq c_i$, and $\mathcal{C} = (c_i, \mathcal{C}_{-i})$. Respecting this relation guarantees that each user prefers to truthfully report its correct cost instead of misreporting its cost, given that all other users are truthful.

4.2.3 Individual rationality

A mechanism is called *individual rational* (IR), if for each user u_i and $c_i \in C_i$, we have:

$$\mathbb{E}_{\mathcal{C}_{-i}}[\mathcal{U}_i(\mathcal{C})] \geq 0\,, \quad i.e. \quad \mathbb{E}_{\mathcal{C}_{-i}}[p_i(\mathcal{C}) - c_i \cdot l_i(\mathcal{C})] \tag{11}$$

Individual rationality expresses that at the Bayesian Nash equilibrium, in truthful reporting strategy of users, each user obtain at least as much utility as the one of not participating at all. In the latter case it is assumed that participation cost and payment are zero.

4.3 Problem Statement

The service provider needs to design a mechanism for participation level and payment such that its expected expenses for reimbursing participants is minimized. The challenges for this mechanism design are as follows [19]:

1) Each user strategically try to maximize its own utility, i.e. the amount of reimbursement minus the participation cost.

2) The service provider is unaware of the actual costs of users.

3) The service provider needs to consider the different quality of information provided by users.

4) The service provider needs to deliver the given expected quality to the users.

Let $\mathcal{M}(\mathcal{C})$ be the space of all mechanisms $M(\mathcal{C})$ which respects the following properties:

- P_1 : The vector $L(\mathcal{C})$ respects the minimum threshold, i.e. $g(L(t), Q(t)) \geq \theta_Q$.
- P_2 : $M(\mathcal{C})$ is incentive-compatible (IC).
- P_3 : $M(\mathcal{C})$ is individually rational (IR).

The problem that the service provider requires to address is the following:

$$\min_{M(\mathcal{C}) \in \mathcal{M}(\mathcal{C})} \mathbb{E}_{\mathcal{C}}\{\sum_{u_i \in \mathcal{N}} p_i(\mathcal{C})\} \tag{12}$$

For each user u_i, let c_i be its true cost and y_i be the declared one. Define $L_i(y_i)$ to be expected allocated participation level to user u_i, if u_i declares its cost as y_i while other users declare their true costs. Hence, we have:

$$L_i(y_i) = \mathbb{E}_{\mathcal{C}_{-i}}[l_i(y_i, \mathcal{C}_{-i})] \tag{13}$$

Let $P_i(y_i)$ be the expected compensation to u_i, if she declares cost y_i and the other users declare true costs, i.e.

$$P_i(y_i) = \mathbb{E}_{\mathcal{C}_{-i}}[p_i(y_i, \mathcal{C}_{-i})] \tag{14}$$

Moreover, suppose $U_i(y_i, c_i)$ be the expected utility for user u_i if she declares cost y_i instead of true cost

c_i, i.e.

$$U_i(y_i, c_i) = P_i(y_i) - c_i L(y_i) \tag{15}$$

Then, the condition for incentive-compatibility is as the following:

$$U_i(c_i, c_i) \geq U_i(y_i, c_i) \Leftrightarrow P_i(c_i) - c_i L(c_i) \geq P_i(y_i) - c_i L(y_i) \tag{16}$$

and the condition for individual rationality would be the following:

$$U_i(c_i, c_i) \geq 0 \Leftrightarrow P_i(c_i) - c_i L(c_i) \geq 0 \tag{17}$$

Theorem 4.1. *[19] A mechanism $M(\mathcal{C}) = (L(\mathcal{C}), P(\mathcal{C}))$ is IC and IR if and only if for all users u_i the following are hold: 1) $L_i(y_i)$ is non increasing on y_i, and 2) the following relation satisfies:*

$$P_i(y_i) = D_i + y_i L_i(y_i) + \int_{y_i}^{\bar{c}_i} L_i(x)dx \tag{18}$$

where $D_i = U_i(\bar{c}_i, \bar{c}_i) = P_i(\bar{c}_i) - \bar{c}_i L_i(\bar{c}_i) \geq 0$, and $\bar{c}_i$ is the upper limit of the support set of cost.

A mechanism $M(\mathcal{C})$ with $D_i = 0$ which minimizes the following relation:

$$\begin{aligned} &\int_{\mathcal{C}} \sum_{i \in N} [l_i(\mathcal{C})(c_i + \frac{H_i(c_i)}{h_i(c_i)})] h(\mathcal{C}) d\mathcal{C} \\ &= \sum_{i \in N} \mathbb{E}_{\mathcal{C}}[L_i(\mathcal{C})(c_i + \frac{H_i(c_i)}{f_i(c_i)})] \end{aligned} \tag{19}$$

and satisfies the properties P_1, P_2, and P_3, it solves optimally the relation in 12 subject to threshold θ_Q, and it is IC and IR.

For given the cost vector $\mathcal{C}$, let the participation level vector $L(\mathcal{C})$ be the solution of the following optimization problem:

$$L(\mathcal{C}) = \arg\min_{l} \sum_{u_i \in \mathcal{N}} l_i(c_i + \frac{H_i(c_i)}{h_i(c_i)}) \tag{20}$$

subject to $g(L(t), Q(t)) \geq \theta_Q$.
Moreover, suppose the payment $p_i(\mathcal{C})$ to each user $u_i \in \mathcal{N}$ be as what follows:

$$p_i(\mathcal{C}) = c_i l_i(\mathcal{C}) + \int_{c_i}^{\bar{c}_i} l_i(s, \mathcal{C}_{-i}) ds \tag{21}$$

then we have the following theorem:

Theorem 4.2. *Let $\delta_i(c_i) = c_i + \frac{H_i(c_i)}{h_i(c_i)} > 0$, and assume that $\delta_i(c_i)$ be non-decreasing on c_i.*
(a) if function $g(L)$ can be written as a monotone function of the sum of terms which are linear in l_i for $u_i \in \mathcal{N}$, i.e. we have:

$$g(L) = I(\sum_{u_i \in \mathcal{N}} k_i l_i) \tag{22}$$

where $k_i \in \mathbb{R}$, t hen, the mechanism 20, 21 is incentive-compatible (IC) and individual-rational (IR), and

minimizes the compensation cost of the provider [19].
(b) if function $g(L)$ can be written as sum of concave strictly increasing functions $g_i(l_i)$, $u_i \in \mathcal{N}$, i.e.

$$g(L) = \sum_{u_i \in \mathcal{N}} g_i(l_i) \tag{23}$$

then the mechanism 20, 21 is incentive-compatible (IC) and individual-rational (IR), and minimizes the compensation cost of the provider.

Theorem 4.2 presents two potential forms of function $g(.)$ which could be exploited by the service provider to minimize the cost of payments while motivate users for participation.

4.4 Our Optimal Solution

In this section we propose an incentive-based mechanism for our problem which holds the requirements of Theorem 4.2, and thence guarantees the best trade-off between users' participations and payments.

Suppose that the *probability density function* of cost C_i be a *uniform distribution* on $[c^i_{min}, c^i_{max}]$, i.e. we have:

$$h_i(x) = \begin{cases} \frac{1}{c^i_{max} - c^i_{min}} & c^i_{min} \leq x \leq c^i_{max} \\ 0 & x < c^i_{min} \ or \ x > c^i_{max} \end{cases}$$

with the corresponding *cumulative density function* $H_i(x)$ as the following:

$$H_i(x) = \begin{cases} 0 & x < c^i_{min} \\ \frac{x - c^i_{min}}{c^i_{max} - c^i_{min}} & c^i_{min} \leq x \leq c^i_{max} \\ 1 & x > c^i_{max} \end{cases}$$

then for the amount of cost $c_i \in [c^i_{min}, c^i_{max}]$, we have $\delta_i(c_i) = c_i + \frac{H_i(c_i)}{h_i(c_i)} > 0$, since:

$$\delta_i(c_i) = c_i + \frac{H_i(c_i)}{h_i(c_i)} = c_i + \frac{c_i - c^i_{min}}{c^i_{max} - c^i_{min}} > 0 \tag{24}$$

Moreover $\delta_i(c_i)$ is non-decreasing on c_i, since if we have $c_i < c_j$, then it is resulted from Relation 24 that $\delta_i(c_i) < \delta_j(c_j)$. This means that the second condition of Theorem 4.2 also satisfies.

Afterwards, from the definition of $g(L(t), Q(t))$ in 7, we have:

$$g(L(t), Q(t)) = \frac{1}{n} \sum_{i=1}^{n} l_i \cdot Q_i(t)$$

which by considering $k_i = Q_i(t)$ and I equal to *identity function*, the third condition of Theorem 4.2.(a) holds.

This means that our proposed approach is incentive-compatible, individual-rational, and it minimizes the compensation cost of provider.

5 Experimental Result

To validate our methodology simulative experiments have been run to simulate the interactions among a set of 50 testers participating at the evaluation campaign of 100 apps, 20 of which genuines and 80 malicious. Simulative experiments have been run on a custom simulator, allowing to vary the percentage

of malicious users for each run. Each run for each configuration has been run 1000 times with different seeds, hence reported results are in the form of average and standard deviation. It is worth noting that no Android apps have been effectively run in the simulator, instead the behavior of each app has been set as simulation parameters, specifying the possibility for the malicious code to activate for any of the actions reported in Table 1.

5.1 Simulative Experiments

In the simulation for each evaluation epoch, a random number of testers ranging between 30 and 50 choose to participate to evaluate that specific app. Table 2 summarizes all the parameter values set for the

Table 2: Simulation Parameters

Parameter	Value
Tester Number	50
Apps	100
Genuine Apps	20
Malicious Apps	80
θ_r , $\theta_{\mathscr{R}}$	0.5 , 0.7
Δ_b, Δ_d, Δ_u	0.25 , 0.6 , 0.15
ρ	2
Number of cycles	1000
Attacker percentage	$10\% \leq x \leq 50\%$
Aggressiveness	$10\% \leq x \leq 100\%$

simulator. As discussed experiments have been run varying both the attacker percentage and aggressiveness. This last parameter, in particular, characterizes the behavior of attackers, stating their percentage of malicious reports among all the ones that release during the simulation. The rationale behind this parameter is the possibility of attackers wishing to conceal their actions, presenting some correct reports in order to increase their reputation before performing the attack. All simulations have been run with the four distinct attacker models described in Section 3.5, i.e. all attackers of any simulation behave according to the same model. From all experiments the extracted results are the reputation for good and malicious testers, their revenue and the percentage of the apps correctly classified by D-BRIDEMAID.

5.2 Results

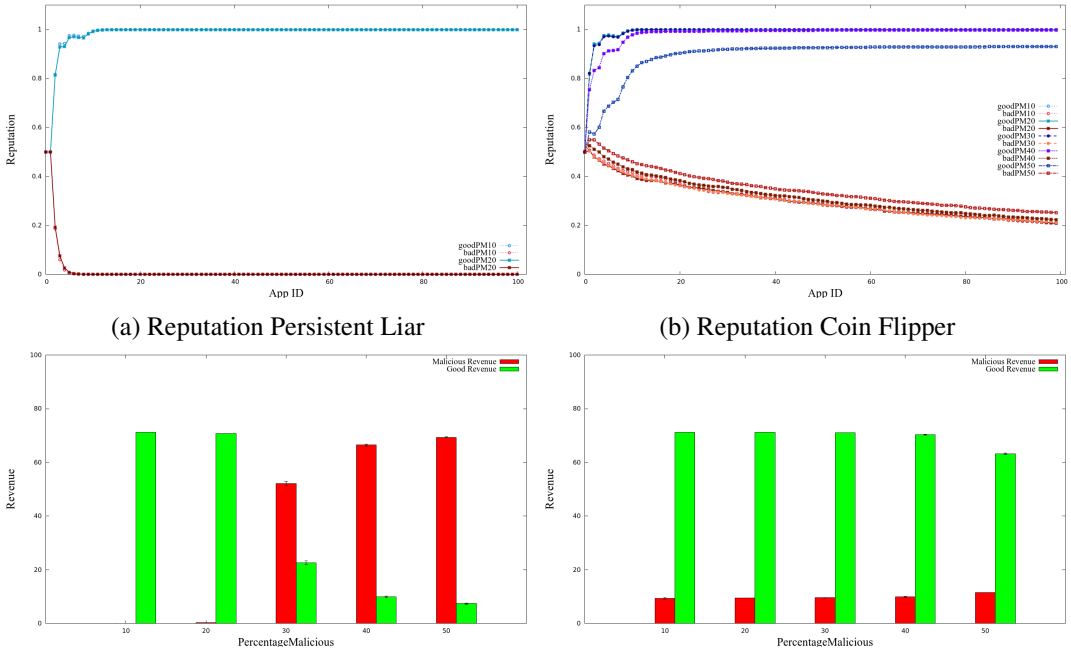
Table 3 schematically reports the classification results of D-BRIDEMAID at the variation of the aforementioned parameters. In particular the table shows the percentage of apps correctly classified, according to the received reports. Results are reported by varying the percentages by 10% at each experiment. As shown for the majority of configurations, the percentage of correctly classified apps is higher than 90%. There is a degradation of the accuracy of the framework only when the malicious tester colludes and they are a consistent percentage of all testers. In particular the attackers are able to defeat the good users exploiting the Persistent Liar attacker model, with a strong aggressiveness (higher than 60%) and when they sum to more than 30% of all users in the system. This is due to the fact that normal users may, provide wrong reports, or reports with a low reliability that are not taken in consideration by the system, whilst the malicious users are colluding to provide a fake report. We recall, in fact, that malicious users always show high reliability in their reports. It is possible to notice a low accuracy also for the case of Malicious App Preacher and Reputation Tamperer. Again this is due to the high reliability given to these

Table 3: Percentage of apps correctly classified.

	Aggressivity	PM 10%	PM 20%	PM 30%	PM 40%	PM 50%
Coin Flipper	10%	100%	100%	100%	100%	100%
	20%	100%	100%	100%	100%	100%
	30%	100%	100%	100%	100%	100%
	40%	100%	100%	100%	100%	99.997%
	50%	100%	100%	100%	99.997%	99.987%
	60%	100%	100%	100%	99.994%	99.95%
	70%	100%	100%	99.995%	99.974%	99.873%
	80%	100%	100%	99.994%	99.94%	99.661%
	90%	100%	100%	99.993%	99.859%	99.059%
	100%	100%	100%	99.977%	99.577%	94.622%
Persistent Liar	10%	100%	100%	100%	100%	100%
	20%	100%	100%	100%	100%	99.996%
	30%	100%	100%	99.999%	99.989%	99.953%
	40%	100%	100%	99.995%	99.944%	99.703%
	50%	100%	100%	99.988%	99.55%	94.626%
	60%	100%	100%	99.766%	90.003%	59.206%
	70%	100%	99.999%	97.181%	56.19%	12.131%
	80%	100%	99.992%	86.102%	16.586%	1.031%
	90%	100%	99.979%	58.813%	2.071%	0.091%
	100%	100%	99.858%	18.109%	0.069%	0.004%
Malicious App Preacher	10%	100%	100%	100%	100%	100%
	20%	100%	100%	100%	100%	99.996%
	30%	100%	100%	100%	99.991%	99.938%
	40%	100%	100%	99.997%	99.945%	99.675%
	50%	100%	100%	99.967%	99.442%	96.703%
	60%	100%	99.999%	99.92%	93.068%	61.658%
	70%	100%	99.999%	98.213%	60.881%	25.813%
	80%	100%	99.988%	88.361%	30.206%	20.29%
	90%	100%	99.984%	60.978%	21.375%	20.002%
	100%	100%	99.863%	31.441%	20.003%	20%
Reputation Tamperer	10%	100%	100%	100%	100%	100%
	20%	100%	100%	100%	100%	100%
	30%	100%	100%	100%	100%	100%
	40%	100%	100%	100%	99.998%	99.991%
	50%	100%	100%	99.998%	99.986%	99.925%
	60%	100%	100%	99.991%	99.917%	99.524%
	70%	100%	100%	99.97%	99.597%	98.053%
	80%	100%	99.994%	99.904%	98.759%	93.961%
	90%	100%	99.992%	99.745%	96.413%	86.861%
	100%	100%	99.99%	99.282%	91.394%	81.168%

reports which become more influential in the case of malicious app, that, by configuration, are 80% of the share of all apps. It must be noticed, however, that the reported performance support the validity of the proposed model, which in the worst setting, is able to be resilient up to 20% of malicious and colluding attackers.

Results for reputation and revenue with different attacker models are reported in Figure 3. In Figure 3a it is represented the average reputation pattern for good and malicious testers, on the evaluation of 100 apps, where all attackers follow the persistent liar model. For the sake of clarity have been reported only the results related to the experiments with 10% and 20% as percentage of malicious users. The reputation results confirm what shown in Table 3, in fact, it is possible to see that reputation of malicious users immediately drops under the threshold, thus not giving the possibility to attackers to send reports considered in the evaluation process. Concerning the revenue (Figure 3c) it is possible to see that attackers do not receive any revenue for the first two experiments, whilst they become able to control the system, thanks to collusion if they are more than 30% of selected testers. Figure 3b and Figure 3d shows reputation and revenue when the attackers use the coin flipper attack model. As shown, the reputation of coin flipper always remains for most of the time under the threshold of 0.5, for all percentages up to 50%. The earned revenue is thus negligible and the system is slightly affected even when the percentage of attackers reaches 50%.



(a) Reputation Persistent Liar

(b) Reputation Coin Flipper

(c) Revenue Persistent Liar

(d) Revenue Coin Flipper

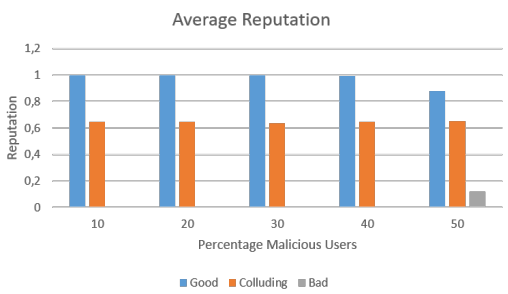
Figure 3: Revenue and reputation for different attacker models.

5.3 Collusion Management

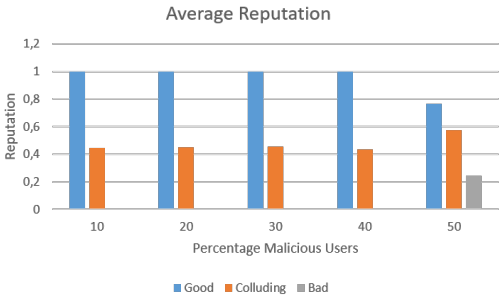
As a further validation of the proposed framework, in addition to the four considered attack models, we have run an additional set of experiments including the presence of colluding attackers. In the former analysis, we have, in fact, considered malicious users acting according to a behavioral pattern as standalone, without any possibility of communication with other malicious users. Colluding attackers can be more dangerous, since they can coordinate define strategies to cause the higher damage to the system. In particular, we will consider a set of attackers which, in order to deceive the reputation system of D-BRIDEMAID, will behave correctly, i.e. as a genuine user, in order to raise their reputation. Hence, when their reputation is over threshold, they will perform a reputation tampering or an app-preaching attack all together. More specifically, in our attack model colluders will not perform any attack if their reputation is under the threshold. After reaching the threshold, if all other colluders have a score higher than the threshold, they will perform the attack.

The experiments have been run with the same configuration of the experiments without collusion, for what concerns the number of apps, users and epoch duration. The percentage of genuine users is varied up to 50%. Of the malicious users, half are colluders, and the remaining half are malicious with an aggressiveness of 1 (i.e. will always misbehave). Two set of experiments have been run, for two different attack models: reputation tamperer and attack preacher. The analyzed apps are 75% genuine and 25% malicious. Figure 4 reports the average revenue and reputation for genuine (good) users, colluders and simply malicious attackers.

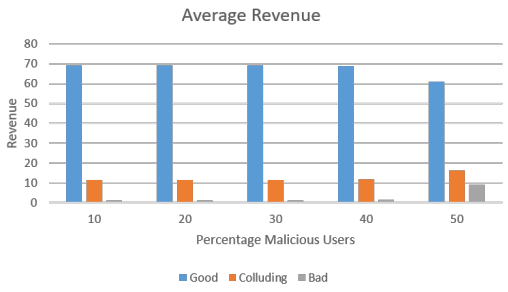
As shown, D-BRIDEMAID prove to be effective minimizing reputation and revenue of colluders and malicious users for both attack types. The colluders manage to have a higher revenue and reputation compared to normal attackers, due to the time spent trying to raise the reputation. Hence, the system will provide revenue only for correct report, either if they are received from genuine or malicious users. Finally Figure 5 shows the evolution over time of the reputation in average of genuine, colluder and



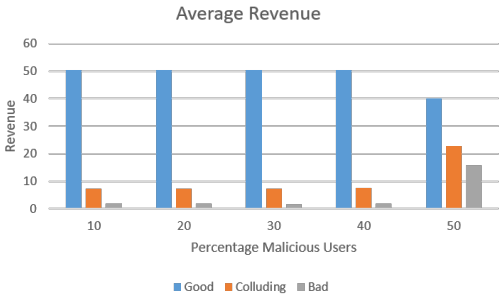
(a) Reputation Preacher



(b) Reputation Tamperer

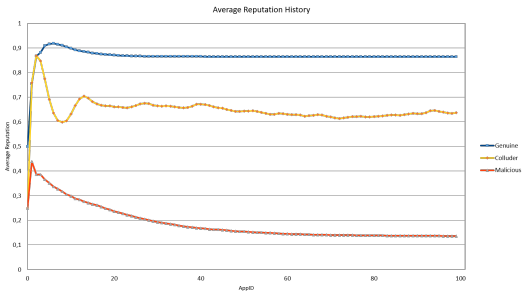


(c) Revenue Preacher

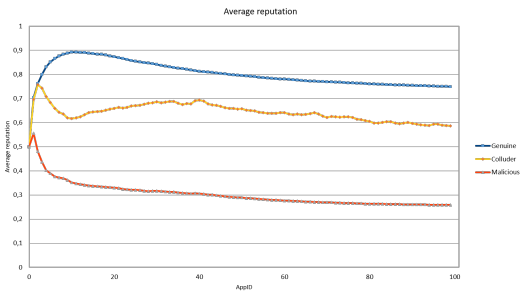


(d) Revenue Tamperer

Figure 4: Revenue and reputation for different attacker models with collusion.



(a) Reputation History Persistent Liar



(b) Reputation History Coin Flipper

Figure 5: Revenue and reputation for different attacker models with collusion.

malicious users. It is worth noting how the reputation of colluders is always around the threshold as a result of their behavior.

6 The Real-World Case Study

In this section we provide, in order to confirm the results obtained from the simulated environment, a real-world scenario in order to show how the D-BRIDEMAID reputation algorithm is able to discriminate between malicious users (i.e., users that attempt to push into the D-BRIDEMAID framework incorrect results) and trusted ones (i.e., users that push results coherent with the BRIDEMAID report). We evaluate the D-BRIDEMAID effectiveness through 10 volunteer participants that generated BRIDEMAID reports related to well-known malware and trusted Android applications.

User	Sex	Age	Vendor	Model	O.S. version	Smartphone	Tablet
#1	F	20	Samsung	S4 mini	4.4.2	X	
#2	M	54	Samsung	Galaxy J1	4.4.2	X	
#3	F	30	Samsung	Galaxy grand prime	5.1.0	X	
#4	M	28	Samsung	S6	6.0.1	X	
#5	M	24	LG	Nexus 5	6.0.1	X	
#6	F	24	Asus	Zenfone 2	5.1.0	X	
#7	F	50	Asus	Zenpad 7	4.4.2		X
#8	F	14	Samsung	Tab 2	4.4.2		X
#9	M	55	Samsung	Galaxy note 3	4.4.2	X	
#10	M	27	Samsung	S3	4.3	X	

Table 4: Devices involved in the evaluation with owner characterization.

We observed 10 users: the evaluation time window began on *August 25, 2017* and finished on *September 1, 2017*: in this 1-week time windows we asked to the participants to generate to push into the D-BRIDEMAID framework their analysis.

Table 4 provides the details of the observed devices used to evaluate the proposed framework in the real-world: in the evaluation both smartphones and tablets are used as experimental environments.

6.1 Dataset

The considered applications in the case study were obtained from the Drebin dataset [20, 21]: it represents a well known collection of mobile malicious samples used in several scientific works, which includes 179 widespread Android families.

The malicious samples dataset comprise families characterized by different installation method i.e., (i)standalone (mobile applications developed with the aim to include malicious functionalities; (ii) repackaging (mobile applications that embed the malware into a trusted application and (iii) update attack (mobile applications that include an update component able to download the malicious action at runtime).

Malware dataset is also partitioned according to the *malware family*; each family contains samples which have in common several characteristics: the payload installation, the kind of attack and the events that trigger malicious payload [14].

Table 5 shows the analyzed 5 malware families in our malware dataset with the details of the installation types, the kinds of attack and the events which activate the payload.

Table 5: Android malware families with details of the installation method (standalone, repackaging, update), the kind of attack (trojan, botnet) and the events that trigger the malicious payload.

Family	Installation	Attack	Activation
Opfake	r	t	user
Moghava	r	t	boot
DroidKungFu	r	t,b	boot
KMin	s	t	boot
Koler.C	s,u	t,b	user

The malware belonging to *Opfake*, *Moghava*, *DroidKungFu* and *Kmin* families were retrieved from

the Drebin project [20, 21], while we obtained the *Koler.C* samples from a collection[1] of freely available malware belonging to HelDroid dataset and appeared from December 2014 to June 2015.

We briefly describe the malicious payload action for the 5 considered families in the real-world experiment.

1. The *Opfake* applications embed an algorithm able to change shape over time with the aim to evade antimalware technologies. The Opfake malicious payload is triggered by the user i.e., by an UI event, and it is able to send premium text messages. As a matter of fact, these messages are sent when the wapxload.ru/opera/ web page is displayed on the infected device screen once the user open it. The malicious payload application does not read the content of the HTTP response but it is able to enables the JavaScript functionality in the browser and registers a callback method named onJsPrompt.

2. The *Moghava* family emerged from third-party Iranian Android-Markets. Differently from others malicious payload embedded in others widespread families, this mobile malware family was not intended in order to extort money to the victim. Instead, this family represents another sample from the side of politically-motivated hacking (the so-called hacktivism) which modifies images that are stored on the device. Even if these malicious samples do not cause any financial loss to the device user, all the pictures on the smartphone are compromised. The malware is able to perform the harmful action when the smartphone receives a "boot completed" event, i.e. when the user starts the device.

3. The *DroidKungFu* malicious payload is able to add into the infected application a service and a receiver. The receiver is able to be notified when the system terminates the boot (in this way the malicious payload is able to automatically launch the service without user interaction). Basically the samples belonging to this family exhibit following behaviours: (i) silent mobile device rooting, (ii) unlocks all system files and functions, (iii) install itself without any user interatction, while it gathers from the infected devices following data: (i) IMEI number, (ii) phone model, (iii) Android OS version, (iv) network operator and type and (v) User private and sensitive information from the device and from the SD Card memory.

4. The *KMin* malicious payload is a trojan that to obtain root access to the infected devices. It is executed each time that the infected devices are booted (as the DroidKungFu family) and it attempts to download and install other malicious files without the consent of the user.

5. The *Koler.C* family is one of the newest threats in mobile environment: the ransomware. The infections belonging to this emerging threat are able to request of a ransom in order to unlock the infected device. They employ different techniques in order to lock the device and then ask for a payment [22, 23], usually in bitcoin.

   If a user browsing online using an Android device lands on a malicious web page, they will be prompted to download an application. Unlike similar attacks against users browsing on a PC or Mac machine, download of the file is neither silent nor automatic; the user must confirm the download and manually install the application. To encourage download and installation, the ransomware is promoted as providing access to adult materials. On installation the infected application opens a browser page that displays a notice over the Home screen showing that the device has been 'locked due to security violations and all files have been encrypted'. The specific wording of the message varies depending on the user's geographical location. For instance, users in Italy will see an Italian-localized message while users in the UK or US will display English text.

[1] `http://ransom.mobi/`

In order to obtain trusted applications we crawled the Google's official store[2] using an open-source crawler[3]. The downloaded dataset includes 5 samples belonging to different categories available on the market.

The legitimate applications were collected in July 2017.

We analyzed the dataset with the VirusTotal web service[4], able to run 57 antimalware technologies (i.e., Symantec, Avast, Kasperky, McAfee, Panda, and others): the analysis confirmed that the obtained applications were trusted while the malware ones were really malicious.

In addition we evaluate the 10 applications under analysis (the 5 malware and the 5 genuine) using the the BRIDEMAID framework and we highlight that the reports confirm the results obtained by the VirusTotal antimalware about the maliciousness of the malware samples and the trustworthiness of the legitimate ones (as a matter of fact, the aim of the case study is to evaluate the D-BRIDEMAID reputation algorithm, not the BRIDEMAID analysis).

6.2 Results

Each users involved in the evaluation analyzed 10 app (5 malware, one for each considered family and 5 genuine).

In order to evaluate the effectiveness of the D-BRIDEMAID reputation algorithm effectiveness with real-world (user and) devices, we consider in the case study the attack scenarios previously analyzed with the simulated environment: the Reputation Tamperer, the Malicious App Preacher, the Persistent Liar and the Coin Flipper. With regards to the Reputation Tamperer we consider the malicious user that marks as malicious a genuine application, while relating to the Malicious App Preacher we consider the malicious user that push the system in considering as legitimate an application which is malicious. The Persistent Liar is represented by the user that try to maximize the system damage producing always the opposite decision with respect to the BRIDEMAID report. The Coin Flipper attack represents a random decision about the trustworthiness of the application under analysis. We highlight that in a real-world scenario it may happen that a malicious application is reported by the BRIDEMAID framework as genuine: this may happens when the user does not run the application under analysis for a considerable time-window or when the events able to activate the malicious payload (as described in Table I) are not sent by the real user device (in this case the BRIDEMAID generated report will mark the application under analysis as genuine and the user will submit to the D-BRIDEMAID system this result). For instance, when the user analyzes a malware belonging to the Opfake family, whether he/she does not click on the button in order to download the fake version of the Opera browser, the malicious payload will not installed on the device: for this reason the BRIDEMAID report will not reflect the maliciousness of the application under analysis. The same situation can happen when the user evaluates a malware belonging to the DroidkungFu or Kmin family and the BOOT_COMPLETED event was not received by the device in order to activate the malicious payload.

Table 6 shows the results of the case study evaluation, with the details of the decision of each user for each application under analysis. We label the malware applications with M_x where $1 <= x <= 5$ identifies the i-th malware application, while we label the genuine applications with T_x where $1 <= x <= 5$ identifies the i-th trusted application evaluated.

We indicate with M the application under analysis submitted to the D-BRIDEMAID framework by the user as malware, and with T the application submitted by the user to the framework as trusted.

Relating to the perpetrated attacks, we highlight that the user #1 exhibits the Coin Flipper attack behaviour (i.e., he/she gives random decision): as we previously highlighted this behaviour is not always

[2] https://play.google.com/store

[3] https://github.com/liato/android-market-api-py

[4] https://www.virustotal.com/

User/App	M_1	M_2	M_3	M_4	M_5	T_1	T_2	T_3	T_4	T_5
#1	T	M	T	M	T	M	T	M	T	M
#2	T	T	T	T	T	M	M	M	M	M
#3	M	M	M	M	M	M	M	M	M	M
#4	T	T	T	T	T	T	T	T	T	T
#5	M	M	M	M	M	T	T	T	T	T
#6	M	M	M	M	M	T	T	T	T	T
#7	M	M	M	M	M	T	T	T	T	T
#8	M	M	M	M	M	T	T	T	T	T
#9	M	M	M	M	M	T	T	T	T	T
#10	M	M	M	M	M	T	T	T	T	T
D-BRIDEMAID	M	M	M	M	M	T	T	T	T	T

Table 6: Real-world case study results: the last row represent the D-BRIDEMAID final decision about the trustworthiness of the applications under analysis performed by the reputation algorithm, while the other ones the users decision basing on the BRIDEMAID report for each application analysed.

symptomatic of a malicious user, as a matter of fact it can be depending from an insufficient execution time of the application or to the absence of the user interaction and/or system events able to activate the malicious payload. The user #2 exhibits the Persistent Liar attack behaviour: the user #2 submits to the D-BRIDEMAID framework the opposite decision with respect to the BRIDEMAID report. The user #3 behaviour is symptomatic of the Reputation Tamperer attack, as a matter of fact he/she aims at tampering the reputation of the legitimate applications considering them as malicious. The user #4 is performing the opposite behaviour with respect to the user #3: the Malicious App Preacher attack. He/she reports that all the applications under analysis are genuine.

Relating to the user #5, #6, #7, #8, #9 and #10 we observe that the submitted decisions are coherent with regard to the BRIDEMAID reports and this is the reason why we consider these users as trusted ones.

We conclude that in the real-world analysis we take into account 4 malicious users (perpetrating respectively 4 different types of attack) and 6 trusted ones: as shown in Table 6 in the real-world scenario the D-BRIDEMAID reputation algorithm was always able to recognize the (malicious and genuine) applications analyzed, making useless the malicious user actions (confirming the results obtained with the simulation environment).

7 Related Work

In this section we review the current literature related to the malware detection collaborative approach.

Researchers in [24] proposed CloudAV, a system in which end hosts send suspicious files to a central cloud-based antivirus service for scanning by several antimalware software. A threshold approach is considered in order to aggregate feedback from different antimalware. The CloudAV prototype is depicted in [25].

The Social-AV tool [26] consider social collaboration and the hot set concept. The second one states that not all malicious signatures are important in the same way. For instance, some signatures (i.e., the so-called hot set) are more likely to be matched than othres malicious signatures. Social-AV only keeps the hot set of signatures in the main memory, while the whole signature database is distributed among devices belonging to the social group of the device owner.

The RAVE [27] collaborative malware scanning system consider emails that are sent to several agents

for malware scanning. A voting based mechanism is take into account in order to make final decisions. Using RevMatch [28] collaborative malware decisions are made based on labeled malware detection history from participating antimalware. In this case the model is evaluated using malicious samples with the aim to demonstrate that collaborative malware detector are able to improve the accuracy with respect to single antiviruses. The main limitation of these approaches is represented by the usage of antimalware, as matter of fact signature provided by free and commercial antimalware are easily evaded by zero-day attacks or using trivial code obfuscation techniques, as demonstrated in [4, 29].

Researchers in [30] consider static analysis on the executables in order to extract function calls in Android applications using the readelf command. Function call are compared with malicious samples in order to classify them using PART i.e., Prism and Nearest Neighbor Algorithms. Our method performs a deeper scan of the application under analysis extracting features using static and dynamic analysis, i.e. BRIDEMAID runs the application in order to have more chances to stimulate the malicious payload.

Researchers in [31] designed a scalable mobile malware detection mechanism using multifeature collaborative decision fusion. The considered features are the permissions and the API calls with the aim to provide a detection by training several classifiers and combine the decisions using collaborative approach (based on probability theory). Their approach is able to reach a precision of 0.989 and a recall of 0.98 evaluating 1073 malicious files belonging to the Contagio project and 904 legitimate applications. Our approach consider a more recent dataset of application, obtaining a precision and a recall higher.

Authors in [32] design an approach for monitoring variations on a mobile device with the aim to detect anomalies. The variations can be caused for instance, by malicious software and attackers (e.g., flooding or network probing). The data monitored for this analysis are sent to a remote server able to generate profiles of each monitored mobile device. The main difference with our method is represented by the fact that this approach focused on network traffic (by WiFi, bluetooth) to detect anomalies and it is applied to devices with Windows Mobile on board.

Researchers in [33] design a malware-detection framework able to monitors, detects, and analyzes unknown threats. The designed framework is composed of a power monitor able to builds a power consumption history from the analysed samples, and a data analyzer which generates a signature from the constructed history. D-BRIDEMAID is able to extract a more extended set of features not only energy-related in order to perform malware identification.

Miettinen et al. [34] propose framework for intrusion detection, which consideres host and network-based detection. Whether an anomaly is detected, the device sends an intrusion alert to a server. The server is able to collect information from sensors with the aim to generate network related intrusion alarms. The researchers consider also a correlation engine with the aim to correlate the device and network intrusion alarms. D-BRIDEMAID extracts a set of features, performing dynamic and static analysis, not only related on network traffic in order to perform a more accurate analysis.

Colajanni et al. [35] propose an architecture with the aim to automate malware collection and classification. Their architecture considers the cooperation of several sensors distributed over several networks. The designed architecture is scalable because the number of component tiers can be adapted to the network characteristics . Furthremore, their method is general and takes advantage of the knowledge of each participant on its network part.

Authors in [36] discuss an anti-obfuscation and collaborative malware detector. Basically the proposed solution identifies the program that behaves suspiciously in end-hosts and between a group of suspicious programs. In order to represet the program behaviour they consider the Handle dependences and Probabilistic Ordering Dependence technology.

NetBuckler [37] is a client application that employs collaborative intelligence in order to detect Internet worms. NetBuckler is able to create a peer-to-peer network able to meet in custom peer groups and communicate traffic information. Their final aim is to enforce security measures depending on the information received.

Lu et al. [38] present CCS, a collaborative online malware analysis framework. Each sensors in the framework analysis is able to analyse malicious samples: CCS is able to aggregate those analyses between different sensors. Authors implemented a CCS proof-of-concept version and evaluate their approach considering 917 real-world malware samples. Our work is different because from these works because is related to Android environment.

A collaborate mechanism proposal to anticipate Advanced Persistent Threat is described in [39]. The paper presents a design to detect zero day attack using windows function hooking that might help the information security community to detect such malicious attacks well in time so the appropriate defensive actions could be taken. This method is able to intercept the invocation of malicious DLLs relating to Microsoft Windows environment, while our method is related to Android OS.

Sakib et al. [40] propose a combination of three models aimed to to capture the output of different antimalware scanners. The adoption of the three models help to predict the accuracy level of each antimalware combination to determine the optimal configuration of the multi-scanner detection system in order to maximise the detection accuracy.

Belaoued et al. [41] present MACoMal, a decision mechanism aimed to assist antimalware to collaborate with each other with the aim to reach a consensual decision about the maliciousness of the submitted suspicious applications.

A collaborative intrusion detection aimed to detect malicious behaviour on IoT and Cloud Networks is discussed in [42]. The Bidirectional Long Short-Term Memory deep learning algorithm is employed for detecting anomalies by analysing network data. The idea behind this method is to propose a decision support system aimed to help cloud providers for secure migration.

Another collaborative intrusion detection framework is presented in [43]. Authors consider machine learning, by applying disagreement-based semi-supervised learning algorithm and evaluating the proposed framework by exploiting real IoT network environments, demonstrating the false alarm reduction obtained by the proposed method.

Li and colleagues [44] adopt the blockchain paradigm to design a framework that aims to combine blockchains with challenge-based trust mechanism. Authors evaluated the proposed method considering random poisoning attack.

A distributed collaborative intrusion detection system for the detection of attacks focused on VANET network is discussed in [45]. Basically this intrusion detection exploit a distributed collaborative detection framework aimed to implement the storage and computation of big data and the tracking of information collection.

8 Conclusion and Future Work

Detecting new threats for mobile devices, is a challenging task which could benefit from cooperation of several users, who actively discover and communicate app anomalies which might represent malicious behaviors. Thus, in this work we have proposed D-BRIDEMAID, which is a distributed collaborative framework for detecting malicious apps for Android devices. We have discussed the framework which is based on the accurate and efficient BRIDEMAID framework for malware analysis, including mechanisms for distributed trust, resilient to different kind of attacks. Also a game theoretical analysis for proposing an incentive mechanism, designed to incentivize user participation is presented, with a set of simulated and real experiments that demonstrate the viability and effectiveness of the proposed approach. The presented framework can be integrated, for increased accuracy with commercial antivirus engines, or Google native security services, reporting also directly reports on apps deemed malicious by the user community. We argue that this should strongly improve the effectiveness of tools such as the Google Play Bouncer service.

Acknowledgements

This work has been partially supported by H2020 EU-funded projects NeCS and C3ISP and EIT-Digital Project HII.

References

[1] MCAfee, "Mobile threat report, what's on the horizon for 2016," https://www.mcafee.com/blogs/consumer/mobile-threats-report-whats-on-the-horizon-for-2016/ [Online; accessed on September 15, 2020], March 2016.

[2] Kaspersky, "Smartphone security: Android vs. iphone vs. blackberry vs. windows phone - top mobile threats," https://usa.kaspersky.com/resource-center/threats/android-vs-iphone-mobile-security [Online; accessed on September 15, 2020], 2016.

[3] CyberSecurityAviationSOC, "Ever-evolving threats," https://airbus-cyber-security.com/wp-content/uploads/2017/10/Airbus_CyberSecurity_SITA_Service-Brochure.pdf [Online; accessed on September 15, 2020], 2017.

[4] G. Canfora, F. Mercaldo, G. Moriano, and C. A. Visaggio, "Composition-malware: building android malware at run time," in *Proc. of the 10th International Conference on Availability, Reliability and Security (ARES'15),Toulouse, France*. IEEE, August 2015, pp. 318–326.

[5] F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, "Download malware? no, thanks. how formal methods can block update attacks," in *Proc. of the 4th IEEE/ACM FME Workshop on Formal Methods in Software Engineering (FormaliSE'16), Austin, Texas, USA*. IEEE, May 2016, pp. 22–28.

[6] F. Martinelli, F. Mercaldo, A. Saracino, and V. C. A., "I find your behavior disturbing: Static and dynamic app behavioral analysis for detection of android malware," in *Proc. of the 14th IEEE Annual Conference on Privacy Security and Trust (PST'16), Auckland, New Zealand*. IEEE, December 2016, pp. 129–136.

[7] F. Martinelli, F. Mercaldo, and A. Saracino, "Bridemaid: An hybrid tool for accurate detection of android malware," in *Proc. of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS'17), Abu Dhabi, United Arab Emirates*. ACM, April 2017, pp. 899–901.

[8] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–974, January-February 2016.

[9] J. Gajrani, J. Sarswat, M. Tripathi, V. Laxmi, M. S. Gaur, and M. Conti, "A robust dynamic analysis system preventing sandbox detection by android malware," in *Proc. of the 8th International Conference on Security of Information and Networks (SIN'15), Sochi, Russia*. ACM, September 2015, pp. 290–295.

[10] M. Faiella, F. Martinelli, P. Mori, A. Saracino, and M. Sheikhalishahi, "Collaborative attribute retrieval in environment with faulty attribute managers," in *Proc. of the 11th International Conference on Availability, Reliability and Security (ARES'16), Salzburg, Austria*. IEEE, August 2016, pp. 296–303.

[11] M. Faiella, A. L. Marra, F. Martinelli, F. Mercaldo, A. Saracino, and M. Sheikhalishahi, "A distributed framework for collaborative and dynamic analysis of android malware," in *Proc. of the 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP'17), St. Petersburg, Russia*. IEEE, March 2017, pp. 321–328.

[12] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Effectiveness of opcode ngrams for detection of multi family android malware," in *Proc. of the 10th International Conference on Availability, Reliability and Security (ARES'15),Toulouse, France*. IEEE, August 2015, pp. 333–340.

[13] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra, "Evaluating the trust of android applications through an adaptive and distributed multi-criteria approach," in *Proc. of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom'13), Melbourne, Victoria, Australia*. IEEE, July 2013, pp. 1541–1546.

[14] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proc. of the 2012 IEEE Symposium on Security and Privacy (S&P'12), San Francisco, California, USA*. IEEE, May 2012, pp. 95–109.

[15] A. Josang, "An algebra for assessing trust in certification chains," in *Proc. of the 1999 Network and Distributed System Security Symposium (NDSS'99), San Diego, California, USA*. Internet Society, February 1999, pp. 89–99.

[16] J. C. Harsanyi, "Games with incomplete information played by "bayesian" players, i–iii part i. the basic model," *Management science*, vol. 14, no. 3, pp. 159–182, November 1967.

[17] Y. Hu and C. K. Loo, "A generalized quantum-inspired decision making model for intelligent agent," *The Scientific World Journal*, vol. 2014, March 2014.

[18] S. V. Albrecht, J. W. Crandall, and S. Ramamoorthy, "Belief and truth in hypothesised behaviours," *Artificial Intelligence*, vol. 235, pp. 63–94, June 2016.

[19] I. Koutsopoulos, "Optimal incentive-driven design of participatory sensing systems," in *Proc. of the 32nd IEEE International Conference on Computer Communications (INFOCOM'13),Turin, Italy*. IEEE, April 2013, pp. 1402–1410.

[20] D. Arp, M. Spreitzenbarth, M. Huebner, H. Gascon, and K. Rieck, "Drebin: Efficient and explainable detection of android malware in your pocket," in *Proc. of the 21th Annual Network and Distributed System Security Symposium (NDSS'14), San Diego, California, USA*. Internet Society, February 2014.

[21] M. Spreitzenbarth, F. Echtler, T. Schreck, F. C. Freling, and J. Hoffmann, "Mobilesandbox: Looking deeper into android applications," in *Proc. of the 28th Annual ACM Symposium on Applied Computing (SAC'13), Coimbra, Portugal*. ACM, March 2013, pp. 808—-1815.

[22] F. Mercaldo, V. Nardone, and A. Santone, "Ransomware inside out," in *Proc. of the 11th International Conference on Availability, Reliability and Security (ARES'16), Salzburg, Austria*. IEEE, September 2016, pp. 628–637.

[23] F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, "Ransomware steals your phone. formal methods rescue it," in *Proc. of the 36th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE'16), Heraklion, Crete, Greece*, ser. Lecture Notes in Computer Science, vol. 9688. Springer, May 2016, pp. 212–221.

[24] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud." in *Proc. of the 17th USENIX Security Symposium San Jose, California, USA*, July-August 2008, pp. 91–106.

[25] C. A. Martínez, G. I. Echeverri, and A. G. C. Sanz, "Malware detection based on cloud computing integrating intrusion ontology representation," in *Proc. of the 2010 IEEE Latin-American Conference on Communications (LATINCOM'10), Bogota, Colombia*. IEEE, September 2010, pp. 1–6.

[26] L. Yang, V. Ganapathy, and L. Iftode, "Enhancing mobile malware detection with social collaboration," in *Proc. of the Third IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT'11) and 2011 IEEE Third International Conference on Social Computing (SocialCom'11), Boston, Maryland, USA*. IEEE, October 2011, pp. 572–576.

[27] C. Silva, P. Sousa, and P. Veríssimo, "Rave: Replicated antivirus engine," in *Proc. of the 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W'10), Chicago, Illinois, USA*. IEEE, June 2010, pp. 170–175.

[28] C. J. Fung, D. Y. Lam, and R. Boutaba, "Revmatch: An efficient and robust decision model for collaborative malware detection," in *Proc. of the 2014 IEEE Network Operations and Management Symposium (NOMS'14), Krakow, Poland*. IEEE, May 2014, pp. 1–9.

[29] E. Medvet and F. Mercaldo, "Exploring the usage of topic modeling for android malware static analysis," in *Proc. of the 11th International Conference on Availability, Reliability and Security (ARES'16), Salzburg, Austria*. IEEE, August-September 2016, pp. 609–617.

[30] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, "Static analysis of executables for collaborative malware detection on android," in *Proc, of the 2009 IEEE International Conference on Communications (ICC'09), Dresden, Germany*. IEEE, June 2009, pp. 1–5.

[31] S. Sheen, R. Anitha, and V. Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach," *Neurocomputing*, vol. 151, pp. 905–912, March 2015.

[32] T. K. Buennemeyer, T. M. Nelson, L. M. Clagett, J. P. Dunning, R. C. Marchany, and J. G. Tront, "Mobile device profiling and intrusion detection using smart batteries," in *Proc. of the 41st Annual Hawaii International Conference on System Sciences (HICSS'08), Waikoloa, Hawaii, USA*. IEEE, January 2008, pp. 296–296.

[33] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proc. of the 6th international conference on Mobile systems, applications, and services (MobiSys'08), Breckenridge, Colorado, USA*. ACM, June 2008, pp. 239–252.

[34] M. Miettinen and P. Halonen, "Host-based intrusion detection for advanced mobile devices," in *Proc. of the 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06), Vienna, Austria*, vol. 2. IEEE, April 2006, pp. 72–76.

[35] M. Colajanni, D. Gozzi, and M. Marchetti, "Collaborative architecture for malware detection and analysis," in *Proc. of the IFIP TC11 23rd International Information Security Conference (SEC'08), Milano, Italy*. Springer-Verlag, September 2008, pp. 79–93.

[36] H. Lu, X. Wang, B. Zhao, F. Wang, and J. Su, "Endmal: An anti-obfuscation and collaborative malware detection system using syscall sequences," *Mathematical and Computer Modelling*, vol. 58, no. 5, pp. 1140–1154, September 2013.

[37] C. Christoforidis, V. Vlachos, and I. Androulidakis, "A crowdsourcing approach to protect against novel malware threats," in *Proc. of the 22nd Telecommunications Forum Telfor (TELFOR'14), Belgrade, Serbia*. IEEE, November 2014, pp. 1063–1066.

[38] H. Lu, X. Wang, and J. Su, "Ccs: Collaborative malware clustering and signature generation using malware behavioral analysis," *International Journal of Hybrid Information Technology*, vol. 5, no. 2, pp. 147–152, April 2012.

[39] N. A. S. Mirza, H. Abbas, F. A. Khan, and J. Al Muhtadi, "Anticipating advanced persistent threat (APT) countermeasures using collaborative security mechanisms," in *Proc. of the 2014 International Symposium on Biometrics and Security Technologies (ISBAST'14) , Kuala Lumpur, Malaysia*, August 2014, pp. 129–132.

[40] M. N. Sakib, C.-T. Huang, and Y.-D. Lin, "Maximizing accuracy in multi-scanner malware detection systems," *Computer Networks*, vol. 169, p. 107027, March 2020.

[41] M. Belaoued, A. Derhab, S. Mazouzi, and F. A. Khan, "Macomal: A multi-agent based collaborative mechanism for anti-malware assistance," *IEEE Access*, vol. 8, pp. 14 329–14 343, January 2020.

[42] O. Alkadi, N. Moustafa, B. Turnbull, and K.-K. R. Choo, "A deep blockchain framework-enabled collaborative intrusion detection for protecting iot and cloud networks," *IEEE Internet of Things Journal*, vol. Early Access, pp. 1–1, May 2020.

[43] W. Li, W. Meng, and M. H. Au, "Enhancing collaborative intrusion detection via disagreement-based semi-supervised learning in iot environments," *Journal of Network and Computer Applications*, vol. Volume 161, p. 102631, July 2020.

[44] W. Li, Y. Wang, J. Li, and M. H. Au, "Toward a blockchain-based framework for challenge-based collaborative intrusion detection," *International Journal of Information Security*, pp. 1–13, February 2020.

[45] M. Zhou, L. Han, H. Lu, and C. Fu, "Distributed collaborative intrusion detection system for vehicular ad hoc networks based on invariant," *Computer Networks*, vol. 172, p. 107174, May 2020.

Author Biography

Antonio La Marra received his MEng in Computer Engineering from University of Pisa. He worked as Research Fellow at the National Research Council of Italy (CNR) on usage control in distributed systems, privacy aware data analysis and Android malware detection. Currently he is CEO of Security Forge, a start-up providing cybersecurity software and services.

Fabio Martinelli is Head of Research at Institute of Informatics and Telematics (IIT) of the Italian National Research Council (CNR) where he leads the Trustworthy and Secure Future Internet group. He is co-author of more than two hundreds of papers on international journals and conference/workshop proceedings. His main research interests involve security and privacy in distributed and mobile systems and foundations of security and trust.

Francesco Mercaldo obtained his Ph.D. in 2015 with a dissertation on malware analysis using machine learning techniques. The research areas of Francesco include software testing, verification, and validation, with the emphasis on mobile systems. He worked as post-doctoral researcher at the Institute for Informatics and Telematics, National Research Council of Italy (CNR) in Pisa (Italy). Currently he is involved as lecturer in Database, Mobile Programming (Bachelor Degree) and Ethical Hacking (Master Degree) courses at the University of Molise (Italy).

Andrea Saracino is a Researcher in the Trustworthy and Secure Future Internet group at National Research Council of Italy (CNR). He coauthored more than 50 papers in mobile malware analysis, data usage control, distributed reputation models and behavioral analysis and he has been involved in the activities of EU funded projects on these topics, such as C3ISP (GA n. 700294), NECS (GA n. 675320) and EIT Digital Trusted Cloud Management. He lectured several course on mobile app security, intrusion detection and authentication at both graduate and post graduate level and is co-organizer of international workshop.

Mina Sheikhalishahi received her Ph.D. in Computer Science at Laval University, Canada, in 2016, with a Ph.D. thesis on ”Spam Campaign Detection, Analysis and Formalization” under the supervision of Prof. Mohamed Mejri and Prof. Nadia Tawbi. She joined the ”Security Group” at Institute for Informatics and Telematics of CNR, Italy as post-doc working with Prof. Fabio Martinelli in 2016. She visited the Security Group at Delft University worked with Prof. Zeki Erkin in 2018. Currently, she is a post-doc researcher in Security Group at Eindhoven University of Technology working with Prof. Nicola Zannone.